Planning for millions of NPCs in Real-Time

Prévost Guillaume, Cardon Stéphane, Jacopin Éric
CReC Saint-Cyr - AMSCC
Guer, F-56380
guillaume.prevost22@orange.fr

Cazenave Tristan
LAMSADE
Université Paris-Dauphine - PSL
Paris, F-75775
tristan.cazenave@lamsade.dauphine.fr

Guettier Christophe
Safran Electronics and Defense
Safran
Fougères, F-35300
christophe.guettier@safrangroup.com

***Abstract*—We address the problem of scaling the generation of plans in real-time to control the behaviors of several millions of Non-Player Characters (NPCs) in video-games and virtual worlds. Search-based action planning, introduced in the game F.E.A.R. in 2005, has an exponential time complexity managing at most several tens of NPCs per frame. A close study of the plans generated in first-person shooters shows that: (1) states are vectors of enumerated values, (2) both initial and final states can be totally defined, (3) actions are both post-unique and unary, (4) plans are totally ordered, and (5) actions occur only once in plans. (1) to (5) satisfy the Simplified Action Structure (SAS) linear time planning framework SAS-PU$\mathbb{T}_1$. We strengthen previous claims on this framework saying that the associated linear time algorithm $\mathbb{P}$ is capable of managing several millions of NPCs per frame by testing it on three new realistic benchmarks that are based on commercial video games.**

***Index Terms*—Artificial Intelligence, Action Planning, SAS Planning, Real-Time, Post-unique, Unary, Linear Time Algorithm**

I. Introduction

F.E.A.R. , a first-person shooter (FPS) released in 2005, was the first video game to use planning to generate character behaviors in real time [1], [2]. The success of F.E.A.R. [3] was such that it led to a wide diffusion of the use of planning in FPSes [4]–[7]; this diffusion was notably facilitated by the publication the following year of a development kit (SDK) that contained the planner code [8]. Today, not only is the success of F.E.A.R. still recognized [9], but the biggest productions, reaching millions of players, do not hesitate to use planning and to make it known [10]–[12], and this despite the real time constraints which are more and more demanding.

In 2005, a game engine was running at about 30 frames per second, that is 33 ms for the whole game logic: among graphics, physics and gameplay mechanics, this leaves less than a millisecond for the planner to generate plans for the few characters that called the planner [13]; ten years later, the move to 60 frames per second has only led to the reduction of the available processing budget for planning. And today, to satisfy the processing budget allocated to the planner, studios explicitly limit the number of calls to at most a few dozen [10], [12], [14] despite the increase in hardware performance. Furthermore, the game situations are designed in such a way that the number of characters is also limited, thus reducing the number of calls to the planner. However, the dynamics of the video game market pushes productions to simulate increasingly large universes with, for the moment, tens of thousands of characters in sight [15] and tomorrow millions of them. The results of [16] suggest that GPUs are a potential solution for such universes in cloud gaming; but what about PCs or game consoles for which GPUs are dedicated to graphics?

The time complexity of planning problems depends on the one hand on language restrictions for representing the planning problem [17] and on the other hand on which part of the input is fixed [18]. From the F.E.A.R. SDK code, we can observe that states are vectors of discrete values and that at certain times, such as fights or routine tasks, the actions are fixed and only the initial and final states are part of the input of a problem instance. Furthermore, the vast majority of the actions are unary [19], i.e., they change the value for only one state variable. Finally, the actions are post-unique by type (attack, defense, deletion, ...), i.e. one type of action is the only one to modify a given state variable; this makes it possible to insert in the plan a generic defense action type, for example, and to delay the choice of the type of defense at the time of the plan execution.

Answers to a questionnaire filled out by several game AI developers validate that our approach is up to date: actions used in commercial game planners can be modeled using the *Simplified Action Structure* (SAS), with post-uniquess (P) and unariness (U) restrictions. This corresponds to the strong NP-Hard SAS-PU class of problems [20], however. Accessing the in-game planning data analyzed in [21], we observe that plans in first-person shooters have two domain-specific features: (1) they are totally ordered, (2) they have only one occurrence of a given type of action: e.g. only one action for threatening, reloading, taking cover, dodging, etc. These two domain-specific features are taken into account in the class of problems SAS-PU$\mathbb{T}_1$ [22] whose notation $\mathbb{T}_1$ compresses two output restrictions: “Totally ordered” (T) and “Type set isomorphism” (T_1). (T) satisfies (1) while (T_1) satisfies (2). In addition, the instances of these problems are solved by a linear-time algorithm, denoted $\mathbb{P}$.

In this paper, we first recall the SAS-PU$\mathbb{T}_1$ planning framework and describe $\mathbb{P}$ phase by phase. We then show the performance of a C++ implementation of $\mathbb{P}$ on three realistic benchmarks which are SAS-PU$\mathbb{T}_1$ versions of the commercial video games Red Dead Redemption 2 [23], Assassin’s Creed: Origins [24] and Horizon Zero Dawn [25].

TABLE I

HORSE BREEDER'S ACTIONS IN RDR-2.

$\mathcal{A}$	pre	post	prv	Explanation
$a^0_{v_0}$	$v_0 = 1$	$v_0 = 0$	$\langle u, u, u\rangle$	Drop Haystack
$a^1_{v_0}$	$v_0 = 0$	$v_0 = 1$	$\langle u, 0, u\rangle$	Take Haystack
$a^2_{v_0}$	$v_0 = 1$	$v_0 = 2$	$\langle u, 0, u\rangle$	Fill Feeder
$a^0_{v_1}$	$v_1 = 1$	$v_1 = 0$	$\langle u, u, u\rangle$	Drop Bucket
$a^1_{v_1}$	$v_1 = 0$	$v_1 = 1$	$\langle 0, u, u\rangle$	Pick up Bucket
$a^1_{v_2}$	$v_2 = 0$	$v_2 = 1$	$\langle u, 1, u\rangle$	Fill Bucket with Water
$a^2_{v_2}$	$v_2 = 1$	$v_2 = 2$	$\langle u, 1, u\rangle$	Fill Horse Trough

$v_0 : Haystack$, $\mathcal{D}_{v_0} = \{0 : stored, 1 : inHands, 2 : inFeeder\}$.
$v_1 : Bucket$, $\mathcal{D}_{v_1} = \{0 : onTheFloor, 1 : inHands\}$.
$v_2 : Water$, $\mathcal{D}_{v_2} = \{0 : inSource, 1 : inBucket, 2 : inFeeder\}$.

A. Background

We use notations from [26] throughout this paper. The Simplified Action Structure (SAS) represents *states* with a set of state variables $\mathcal{M}$, each of which can take a finite number of discrete values or be *undefined*; we note $\mathcal{D}_{v_i}$ the set of values of a state variable $v_i \in \mathcal{M}$. Three sets of state variables are used to represent the conditions of an *action*: (1) *postconditions* (post) define new values for some state variables, (2) *preconditions* (pre) require specific values for all the state variables that are modified in the postconditions, and (3) *prevail-conditions* (prv) require specific values for some state variables which are neither defined in the preconditions nor in the postconditions. In fact, the prevail-conditions are preconditions that will not be modified by the action. An action is *applicable* in a state iff its pre- and prevail-conditions are consistent with their values in that state; *applying* an action in a state changes the values of all the state variables of its postconditions while its prevail-conditions remain unchanged. In an SAS-structure, an action can only change a state variable from one defined value to another defined one (S6), and no *initial* or *goal* state variable can be undefined (S7). A *plan* is a sequence of actions such that any state resulting from applying an action of the sequence is consistent with the next action in the sequence; a plan solves a *problem instance* made of the initial and goal states $(s_0, s_\star)$ and a set of action types iff (1) any action in the plan is a distinct instance of an *action type* of the planning problem, (2) the first action is applicable in s_0, (3) and applying the last action of the sequence *results* in $s_\star$.

Presently, commercial video games that have implemented a planner to control their Non-Playable Characters (NPCs) in real-time [27] are facing the general intractability of planning [28]. They apply restrictions to frame their planner and to respect the processing budget, but their planning problems remain intractable [21], [29]. The approach of [26] and [22] to break the intractability is to use restrictions to frame the planning problems, instead, in order to create tractable classes of problems. In particular, the class of problems SAS-PU$\mathbb{T}_1$ [22], described in the next subsection, is relevant to our framework.

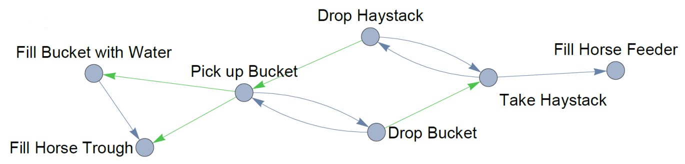

Fig. 1. The orders between the `Horse Breeder`'s action types. The blue directed edges represent the post-pre dependencies while the green ones represent the post-prv dependencies.

B. The Class of Problems: SAS-PU$\mathbb{T}_1$

The class of problems SAS-PU$\mathbb{T}_1$ is composed of the restrictions: (*Post-uniqueness*) no two distinct action types can change the same state variable to the same value (P), (*Unaryness*) each action type changes the value of exactly one state variable (U), (*Totally ordered*) the action plan is totally ordered (T) and (*Type set Isomorphism*) the same action does not occur twice in the action plan (T_1). The restrictions (P) and (U) are input restrictions while (T) and (T_1) are output restrictions.

Tab. I and Fig. 1 present a problem of this class: the `Horse Breeder`, which is based on the video game Red Dead Redemption 2 (RDR-2). The actions are unary because their post-condition affects only one state variable, and they are post-unique because the same post-condition does not occur twice. Due to these two restrictions, (P) and (U), each action is identifiable with the pair (v_i, p). We thus identify them with the format $a^p_{v_i}$ as it can be seen in the column "$\mathcal{A}$" of Tab. I. This problem is (T_1): for any instance (s_0, $s_\star$) of the problem, the minimal solution plan, i.e. the solution plan having the least number of actions between s_0 and $s_\star$, does not contain the same action twice. Eventually, the action types of the problem are partially-ordered, but any partial-ordered plan can easily be totally ordered with a topological sort [30] so as to respect the output restriction (T).

C. Linear time algorithm: $\mathbb{P}$

Any SAS-PU$\mathbb{T}_1$ problem instance can be solved by a correct and complete linear-time algorithm denoted $\mathbb{P}$ [22]. $\mathbb{P}$ benefits from a pre-processing that takes advantage of (P) and (U) to store actions in a hashing table. During this pre-processing, an *identifier*, two *predecessor sets*, and two getters-setters of possible *successors* are assigned to each action. $\mathbb{P}$ plans backwards using the identifiers and the predecessor sets to find a minimal and totally ordered solution plan solving a problem instance formed by s_0 and $s_\star$. To do so, $\mathbb{P}$ works in 3 phases:

Phase 1: For each $v_i \in \mathcal{M}$, if $s_0[v_i] \neq s_\star[v_i]$, then $\mathbb{P}$ builds a *chain of actions* on v_i between $s_0[v_i]$ and $s_\star[v_i]$. A chain of actions on v_i is a sequence of post-unique and unary actions totally ordered with respect to their *post-pre dependency* only. For example with Tab. I, $\langle a^1_{v_0}, a^2_{v_0}\rangle$ is a chain of actions from $v_0 = 0$ to $v_0 = 2$, and $\langle a^1_{v_2}, a^2_{v_2}\rangle$ is a chain of actions from $v_2 = 0$ and $v_2 = 2$. At the end of phase 1, we have a partial-ordered plan $(\mathcal{D}, \preceq_{pre})$, with $\mathcal{D}$ the set of actions that are part of a chain and a partial-order $\preceq_{pre}$ that represents the post-pre dependencies between the actions of $\mathcal{D}$.

Phase 2: $\mathbb{P}$ checks the prevail-conditions of each action a of $\mathcal{D}$ using the set of predecessors of a having a *post-prv dependency*. For each predecessor $pred_{prv}(a)$, if $pred_{prv}(a)$ is already in $\mathcal{D}$, then $pred_{prv}(a)$ is simply ordered before a with the partial-order $\preceq_{prv}$: $pred(a) \preceq_{prv} a$. A successor of the predecessor by *post-pre dependency* $succ_{pre}(pred_{prv}(a))$ may threaten the prevail-condition of a, in that case a must be ordered before $succ_{pre}(pred_{prv}(a))$: $a \preceq_{threat} succ_{pre}(pred_{prv}(a))$. If $pred_{prv}(a)$ is not in $\mathcal{D}$, then $\mathbb{P}$ builds a chain of actions between the start and the unsatisfied prevail-condition of a. The last action of this chain will be $pred_{prv}(a)$ and it is ordered before a. Then $\mathbb{P}$ builds a chain of actions between the unsatisfied prevail-condition of a and the start. The first action of this chain, if not empty, is the successor of $pred_{prv}(a)$ and threatens the prevail-condition of a, therefore a is ordered before $succ_{pre}(pred_{prv}(a))$. The actions of the two newly built chains are added to $\mathcal{D}$ and their prevail-conditions must be checked as well. At the end of phase 2, we have a correct and minimal partial-ordered plan $(\mathcal{D}, \preceq)$, with $\mathcal{D}$ the set of actions found during phase 1 and 2, and $\preceq$ a partial-order grouping $\preceq_{pre}$, $\preceq_{prv}$ and $\preceq_{threat}$.

Phase 3: $\mathbb{P}$ topologically sorts $(\mathcal{D}, \preceq)$ to return $(\mathcal{D}, \prec)$, a minimal and totally-ordered action plan with $\prec$ the total-order between the actions of $\mathcal{D}$.

For the details of the algorithm, the reader shall refer to [22]. Let consider the instance $(s_0 = \langle 0,0,0\rangle, s_\star = \langle 0,0,2\rangle)$ of the `Horse Breeder` problem (Tab. I). During phase 1, $\mathbb{P}$ builds only one chain of actions: $\langle a^1_{v_2}, a^2_{v_2}\rangle$, because $s_0[v_2] \neq s_\star[v_2]$. We have $\mathcal{D} = \{a^1_{v_2}, a^2_{v_2}\}$. During phase 2, $a^1_{v_2}$ has one predecessor by post-prv dependency: $a^1_{v_1}$, which is not in $\mathcal{D}$. Therefore, $\mathbb{P}$ builds a chain from $s_0[v_1]$ to $prv(a^1_{v_2})[v_1]$: $\langle a^1_{v_1}\rangle$. The last action of the chain is $a^1_{v_1}$ and it is ordered before $a^1_{v_2}$: $a^1_{v_1} \preceq_{prv} a^1_{v_2}$. Then $\mathbb{P}$ builds the chain from $prv(a^1_{v_2})[v_1]$ to $s_0[v_1]$: $\langle a^0_{v_1}\rangle$. $a^0_{v_1}$ is the first action of the chain and is a successor of $a^1_{v_1}$. $a^0_{v_1}$ threatens $prv(a^1_{v_2})[v_1]$, therefore: $a^1_{v_2} \preceq_{threat} a^0_{v_1}$. This order ensures $a^1_{v_2}$ will be topologically sorted before $a^0_{v_1}$ during phase 3. $a^0_{v_1}$ and $a^1_{v_1}$ are added to $\mathcal{D}$. Then, $a^2_{v_2} \in \mathcal{D}$ has the same prevail-condition as $a^1_{v_2}$, thus $a^0_{v_2}$ is ordered after $a^1_{v_1}$ and before $a^0_{v_1}$ like $a^1_{v_2}$. Finally, the actions $a^0_{v_1}$ and $a^1_{v_1}$ do not have any defined prevail-conditions. Fig. 2 gives a graphical representation of the orders between actions built during phase 1 and phase 2. The blue directed edges represent $\preceq_{pre}$, the green ones $\preceq_{prv}$ and the red ones $\preceq_{threat}$. During phase 3, $\mathbb{P}$ performs a topological sort to return the totally-ordered plan: $\langle a^1_{v_1}, a^1_{v_2}, a^2_{v_2}, a^0_{v_1}\rangle$, which is the minimal solution plan solving the problem instance.

To summarize this example, the `Horse Breeder` needs the bucket in hands to fill the bucket then the horse trough, and the threat is that the `Horse Breeder` drops the bucket before any filling actions, which would obviously not be correct.

II. Realistic Benchmarks and Results

In this section we introduce 3 concrete benchmarks, represented as SAS-PU$\mathbb{T}_1$ problems, that we created inspired by the following commercial video games: Red Dead Redemption 2 (RDR-2) [23], Assassin's Creed: Origins (ACO) [24] and Horizon Zero Dawn (HZD) [25]. The benchmarks are named after those of the games. We present the world representation, some action types and possible starts and goals for each game. Each problem instance thus formed is solved with plans of 1 to 10 actions. The section is organized as followed: RDR-2, ACO and HZD are introduced; Then, we present the experiments we carried out to study the performance of $\mathbb{P}$. Finally, we explain the results and conclude.

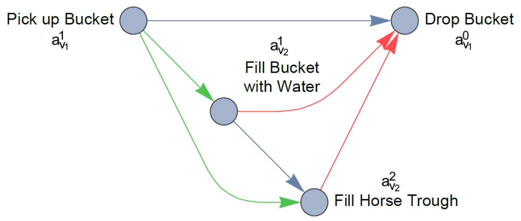

Fig. 2. A graphical representation of the orders between the actions built during phase 1 and phase 2, in order to solve: $(s_0 = \langle 0,0,0\rangle, s_\star = \langle 0,0,2\rangle)$, a `Horse Breeder` problem instance.

A. Realistic Benchmarks

a) Red Dead Redemption 2 (RDR-2): It was released by Rockstar Studios® in October 2018 [23]. It is an action-adventure and open-world video game that takes place in Southern United States in the late Wild West era, in 1899. The player plays as Arthur Morgan, an outlaw member of the Van Der Linde gang who must deal with the decline of the Wild West whilst attempting to survive several different adversaries like the government forces, the rival gangs, etc. This very successful video game [31] has NPCs of great quality which improves credibility and dynamism of the virtual world of RDR-2 [32].

Videos of NPCs' daily life can be found on-line and show the details of their behaviors. At the time stamp 5:36 of the video [33], we follow a `Horse Breeder` that is off to work. He goes to feed his horses by filling the water trough first and then the feeder. Tab. I presents a possible SAS-PU$\mathbb{T}_1$ set of actions for the `Horse Breeder`. Similar to the video, our version of the `Horse Breeder` can interact with a bucket and the haystacks in order to fill the water trough and the feeder for his horses. Then, his work goal is to feed the horses: $s_\star(\text{FeedHorses}) = \langle 2,0,2\rangle$. The main difficulty in this representation is to ensure the NPCs do not carry a haystack and the bucket at the same time as it is physically demanding. Hence the prevail-conditions: $prv(a^1_{v_0})[v_1] = prv(a^2_{v_0})[v_1] = 0$ and $prv(a^1_{v_2})[v_1] = prv(a^2_{v_2})[v_1] = 0$. With the goal $s_\star(\text{FeedHorses})$, it is not possible for the `Horse Breeder` to fill the feeder first, however. The planner will necessary return a plan whose first subgoal is to fill the water trough then the feeder. Because the action Fill Horse Feeder sets $(v_0 = 2)$ permanently, which means the prevail-condition of Pick up Bucket can never be satisfied after FillHorseFeeder while planning. If we want to fill the feeder first anyway, then we need two subgoals: $s_\star(\text{FillFeeder}) = \langle 2, 0, s_0[v_2]\rangle$ and

TABLE II
THE CITIZEN'S ACTIONS IN ACO.

$\mathcal{A}$	Pre	Post	Prv	Explanation
$a^1_{v_0}$	$v_0 = 0$	$v_0 = 1$	$\langle u, u, u, u, u, u, 0\rangle$	BuyFood(?pos)
$a^2_{v_0}$	$v_0 = 1$	$v_0 = 2$	$\langle u, u, u, u, u, u, u\rangle$	Eat
$a^1_{v_1}$	$v_1 = 0$	$v_1 = 1$	$\langle u, u, u, u, u, u, 0\rangle$	BuyDrink(?pos)
$a^2_{v_1}$	$v_1 = 1$	$v_1 = 2$	$\langle u, u, u, u, u, u, u\rangle$	Drink
$a^0_{v_2}$	$v_2 = 1$	$v_2 = 0$	$\langle u, u, u, 0, u, u, 0\rangle$	Store (?item,?pos)
$a^1_{v_2}$	$v_2 = 0$	$v_2 = 1$	$\langle u, u, u, 0, u, u, 0\rangle$	Take (?item,?pos)
$a^0_{v_3}$	$v_3 = 1$	$v_3 = 0$	$\langle u, u, u, u, u, u, u\rangle$	StopWandering
$a^1_{v_3}$	$v_3 = 0$	$v_3 = 1$	$\langle u, 2, u, u, u, u, 0\rangle$	HaveAWalk
$a^0_{v_4}$	$v_4 = 1$	$v_4 = 0$	$\langle u, u, u, u, u, u, u\rangle$	StopDebate
$a^1_{v_4}$	$v_4 = 0$	$v_4 = 1$	$\langle u, u, u, u, u, u, u\rangle$	Debate(?with, ?pos)
$a^1_{v_5}$	$v_5 = 0$	$v_5 = 1$	$\langle u, u, u, 0, 0, u, 0\rangle$	Rest (?pos)
$a^0_{v_6}$	$v_6 = 1$	$v_6 = 0$	$\langle u, u, u, u, u, u, u\rangle$	EndWork
$a^1_{v_6}$	$v_6 = 0$	$v_6 = 1$	$\langle 2, 2, 1, 0, u, u, u\rangle$	Work (?item, ?pos)

v_0 : $Hunger$, $\mathcal{D}_{v_0} = \{0 : noFood, 1 : hasFood, 2 : fed\}$.
v_1 : $Thirst$, $\mathcal{D}_{v_1} = \{0 : noDrink, 1 : hasDrink, 2 : hydrated\}$.
v_2 : $hasItem$, $\mathcal{D}_{v_2} = \{0 : false, 1 : true\}$.
v_3 : $Wandering$, $\mathcal{D}_{v_3} = \{0 : false, 1 : true\}$.
v_4 : $Debating$, $\mathcal{D}_{v_4} = \{0 : false, 1 : true\}$.
v_5 : $isRested$, $\mathcal{D}_{v_5} = \{0 : false, 1 : true\}$.
v_6 : $Working$, $\mathcal{D}_{v_6} = \{0 : false, 1 : true\}$.

$s_\star$(FillTrough) $= \langle s_0[v_0], 1, 2\rangle$. Once the `Horse Breeder` has finished the goal $s_\star$(FillFeeder), then he no longer carries an haystack. Therefore, the achievement of $s_\star$(FillFeeder) set v_0 to the value 0. Same for $s_\star$(FillTrough), once achieved v_2 can be reset to 0 for the `Horse Breeder`. It works the same way with $s_\star$(FeedHorses). In a way, the state variables $HayStack$ and $Water$ can be seen as targetable objects. Once in the feeder, the haystack is no longer targetable. That is, the `Horse Breeder` has to target another one if he wants to repeat the process. This is the same thing for the water, once in the water trough, it is no longer usable and the `Horse Breeder` needs to find a water source to refill the bucket. Hence, if the `Horse Breeder` calls the planner again to $s_\star$(FeedHorses) with new targeted items, then $s_0[v_0] = 0$ and $s_0[v_2] = 0$ necessarily.

Similar to the `Horse Breeder`, we have designed 6 more NPC types for RDR-2: the `Farrier`, whose goals are $s_\star$(Shoeing the horse) and $s_\star$(Forge horseshoes); the `Native hunter`, whose goal is to $s_\star$(Hunt); the `Bandit`, whose goal is to $s_\star$(Rob a bank); the `Innkeeper`, whose goals are to $s_\star$(Serve a drink) and $s_\star$(Assing a room); the `Dancer`, whose goal is to $s_\star$(Dance); and, finally, the `Drunk Citizen`, whose goal is to $s_\star$(Forget). In total, there are 56 actions, 30 state variables and the biggest variable domain has 5 different values.

b) Assassin's Creed: Origins: It is an action role playing video game developed by Ubisoft® Montreal and released in October 2017 [24]. The tenth installment of the Assassin's Creed serie sets in Egypt, near the end of the Ptolemaic period (49–43 BC), where the player plays as a Medjay named Bayek of Siwa. In order to make their magnificent open-world feel more alive and believable, the developers went in great detail with their NPCs, giving them autonomy and freedom through a Goal-Oriented AI system [34]. Soldiers, citizens, and even the fauna, have goals and needs that the player can witness or

TABLE III
ACTIONS OF THE A-M IN HZD.

$\mathcal{A}$	Pre	Post	Prv	Explanation
$a^1_{v_0}$	$v_0 = 0$	$v_0 = 1$	$\langle u, u, u, 1, 1, 0, 0\rangle$	ScanOreDeposit(?from)
$a^1_{v_1}$	$v_1 = 0$	$v_1 = 1$	$\langle 1, u, u, u, u, 0, 0\rangle$	ShareOrePos(?list)
$a^1_{v_2}$	$v_2 = 0$	$v_2 = 1$	$\langle 1, u, u, 1, 1, 0, 0\rangle$	HarvestOre(?type,?pos)
$a^2_{v_2}$	$v_2 = 1$	$v_2 = 2$	$\langle u, u, u, 1, 1, 0, 0\rangle$	RefineOre
$a^3_{v_2}$	$v_2 = 2$	$v_2 = 3$	$\langle u, u, u, 1, 1, 0, 0\rangle$	StoreRefinedOre(?pos)
$a^1_{v_3}$	$v_3 = 0$	$v_3 = 1$	$\langle u, u, u, u, u, 0, 0\rangle$	Rest(?pos)
$a^1_{v_4}$	$v_4 = 0$	$v_4 = 1$	$\langle u, u, u, 1, u, 0, 0\rangle$	Repair
$a^0_{v_5}$	$v_5 = 1$	$v_5 = 0$	$\langle u, u, u, u, u, u, 0\rangle$	ScanDisturbance
$a^0_{v_6}$	$v_6 = 1$	$v_6 = 0$	$\langle u, u, u, u, u, u, u\rangle$	Attack

v_0 : $oreDepositFound$, $\mathcal{D}_{v_0} = \{0 : false, 1 : true\}$.
v_1 : $oreDepositPosShared$, $\mathcal{D}_{v_1} = \{0 : false, 1 : true\}$.
v_2 : $harvestRoutine$,
$\mathcal{D}_{v_2} = \{0 : none, 1 : oreHarvested, 2 : oreRefined, 3 : oreStored\}$.
v_3 : $isCharged$, $\mathcal{D}_{v_3} = \{0 : false, 1 : true\}$.
v_4 : $system$, $\mathcal{D}_{v_4} = \{0 : damaged, 1 : operational\}$.
v_5 : $disturbanceExist$, $\mathcal{D}_{v_5} = \{0 : false, 1 : true\}$.
v_6 : $isThreatened$, $\mathcal{D}_{v_6} = \{0 : false, 1 : true\}$.

feel while playing [35]. As there are a lot of different types of citizens in ACO, we have decided to create a base-NPC named `Citizen`. It is possible to create more specific NPCs that can inherit the citizenship. As an example, the `Horse Breeder` of RDR-2 could be implemented in ACO and it can inherit the actions of the `Citizen`. We gave our `Citizen` the possibility to eat, drink, buy some food, wander, debate and work (Tab. II). Although our action representation is quite generic, we have added some parameters to some of them (BuyFood, BuyDrink, Store, Take, Debate, Rest and Work) so as to allow some contextualization. For instance, if the NPC is a musician who like to play in the street, then we can contextualize: Work(?Luth,?MarketPlace). We can also represent a group of musicians: it is merely a set of NPCs playing different instruments at the same place. We added these parameters in order to illustrate that it is possible to contextualize an action without modifying the overall SAS representation. Furthermore, we want to point out that MoveTo actions must not be represented in any SAS-PU representation. MoveTo actions are related to pathfinding, representing them in any SAS-PU representation is almost impossible as it is very likely to not respect the post-uniqueness. A position to reach can merely be added as a parameter. Indeed, almost every action is preceded by a MoveTo, so it can implicitly be added inside the animation of the action.

For our experiment, we give the `Citizen` one initial state $s_0 = \langle 0, 0, 0, 0, 0, 1, 0\rangle$ which corresponds to the beginning of a day. He has not wandered, debated nor worked yet. He is rested, however, and he is thirsty and hungry. We give him 8 goals such as $s_\star(\text{Work}) = \langle 2, 2, 1, 0, s_0[v_4], 1, 1\rangle$ or $s_\star(\text{BuySupplies}) = \langle 1, 1, s_0[v_2], s_0[v_3], s_0[v_4], s_0[v_5], 0\rangle$ or $s_\star(\text{EatAndDebateWhileWandering}) = \langle 2, 2, s_0[v_2], 1, 1, s_0[v_5], 0\rangle$. During the experiment, `Citizens` are asked to find an action plan from s_0 to one of these goals.

c) Horizon Zero Dawn: It is an action role play game developed by Guerrilla Games® and released in early 2017 [25]. The player plays a huntress named Aloy in a world full of

TABLE IV
RUNTIME PERFORMANCE OF $\mathbb{P}$ WRT. THE LINEAR COMBINATIONS OF THE NUMBER OF NPCS (x) CONSTRUCTED WITH THE SAME DATA AS FIG. 3.

Benchmark	Linear combination of x	$avg(\|\mathcal{R}\|)$
HZD	$0.0676405 + (3.7637 \cdot 10^{-7})x$	12.6667
RDR-2	$0.0313575 + (2.46048 \cdot 10^{-7})x$	8.16667
ACO	$0.0157308 + (1.54132 \cdot 10^{-7})x$	6.6667

high-tech colossal machines that have appeared at the fall of human civilization. These machines are categorized in classes: the *acquisition* class, whose role is to harvest resources; the *combat* class, whose role is to guard vulnerable machines from hunters; the *recon* class, whose initial role was to search for suitable terraforming land, they finally serve as guards and lookouts in the game; the *transport* class, whose role is to help acquisition machines to move large amount of resources. There are other types of machines whose description can be found in [36]. In this paper, we only present a SAS-PU$\mathbb{T}_1$ version of the *acquisition* machines, which we henceforth denote `A-M`. `A-M`s harvest resources of the HZD world. Their role is crucial as they assist the terraforming program and allow for the further construction of machines. This description, however, is more folkloric than anything else as this is not what really happens in-game. Although there exist stunning combat scenes in HZD, which are mainly due to great animations, the overall behavior of the machines can be disappointing to those who seek consistency with the life goal of these machines. Through planning, however, it is possible to give more depth in their behavior to fit their life goals better. Tab. III, for instance, gives a possible SAS-PU$\mathbb{T}_1$ representation for the `A-M`.

In our experiment, the `A-M`s only have one goal: $s_\star(\text{Harvest}) = \langle 1, 1, 3, 1, 1, 0, 0\rangle$. Depending on the initial state, the `A-M` must find an adapted action plan to accomplished its only goal. We defined 6 different initial states for our simulation. For example: $s_0 = \langle 1, 1, 1, 1, 0, 0, 1\rangle$ which means the `A-M` knows some ore positions, these positions are shared with its counterpart and it has already harvested some. The `A-M` is also charged, damaged and under attack. The solution plan in this example is $\langle a^0_{v_6}, a^1_{v_4}, a^2_{v_2}, a^3_{v_2}\rangle$, i.e. the `A-M`'s plan is to fight back, then repair itself before going back to work. Another possible start is $s_0 = \langle 0, 0, 0, 1, 1, 0, 0\rangle$ which states that the `A-M` is ready to work but it has not started yet and does not know the position of ore deposits. The minimal solution plan is $\langle a^1_{v_0}, a^1_{v_1}, a^1_{v_2}, a^2_{v_2}, a^3_{v_2}\rangle$, that is the `A-M`'s plan is to search for ore deposit, share the ore position and, finally, harvest, refine and store the ore.

B. Experiments and Results

The experiments were performed with the following configuration: AMD Ryzen™ 7 2700X (8-Core) CPU (3.7GHz), 32Gb of RAM and Windows 10 (64 bits); $\mathbb{P}$ was written in C++14 with default settings for Microsoft Visual Studio 2019. The question we wish to answer is: "How many NPCs, in RDR-2, ACO or HZD respectively, can get a plan in real time by our C++ implementation of $\mathbb{P}$?". For each benchmark, with respect to their description given in subsection II-A, $\mathbb{P}$ has to provide a minimal solution plan to an increasing number of NPCs as quickly as possible. The goals (or starts) given to the NPCs, among those defined, varied at each call of $\mathbb{P}$. Then, the runtime of $\mathbb{P}$ to return all the requested plans should not exceed $1.67ms$, which represents the real-time constraint of a 60 FPS[1] video game[2]. The results of these experiments are given in Figure 3 and in Table IV. The x-axis of Figure 3 gives the number of NPCs asking for a plan, and the y-axis gives the runtime in millisecond of $\mathbb{P}$ to provide all the requested plans. Each point of the Figure is the average of about 20 repetitions. As an example, the red triangle located at coordinate $(2, 720, 000; 1.0)$ means that the task of $\mathbb{P}$ was to provide $2, 720, 000$ plans in real-time to NPCs in RDR-2, and, after repeating this task 20 times, the average runtime of $\mathbb{P}$ is $1.0ms$. The blue squares represent the HZD data, the red triangles those of RDR-2 and the yellow crosses those of ACO.

Each curve grows linearly and this linearity is explained by the fact the worst-case time complexity of $\mathbb{P}$ is not a function of the number of NPCs. The slope of these curves is merely the ratio of the runtime growth to the number of NPCs growth, which means that the slope highlights the difficulty of $\mathbb{P}$ to solve a problem. The steeper the slope, the more difficult the problem. HZD is thus the hardest problem for $\mathbb{P}$, followed by RDR-2 then ACO. Table IV highlights the causality of this difficulty and gives a value to these slopes. Let $\mathcal{R}$ the set of orders between the actions of the solution plan, Table IV gives the average number of orders ($avg(|\mathcal{R}|)$) of each benchmark. This table also gives the linear regression of each data set, and the argument x refers to the number of NPCs. The linear regressions are the linear functions of the three curves in Figure 3, they give us the slope value of each curve. The results of this table are sorted in descending order of $avg(|\mathcal{R}|)$ and the slopes are sorted equivalently, which entails that the slopes are correlated with the average number of orders between the actions. It is consistent with the worst-case time complexity of $\mathbb{P}$ which is linear in the number of actions in the problem ($|\mathcal{A}|$) plus the number of orders between the actions of $\mathcal{A}$ ($|\mathcal{R}_\mathcal{A}|$): $O(|\mathcal{A}| + |\mathcal{R}_\mathcal{A}|)$ [22]. In other words, the more orders, the more difficult for $\mathbb{P}$. It explains why HZD is the most difficult problems for $\mathbb{P}$ despite being the one with the less state variables and with the less action types. It is in fact more constrained than the other two, and besides, these many constraints are directly linked to the number of the defined prevail-conditions (cf. Table III). Finally, the key result of these data is that our C++ implementation of $\mathbb{P}$ was able to provide a plan of 1 to 10 actions to 3.4 millions of NPCs in less than $1.5ms$, for each of the three realistic benchmarks, which is impressive and definitely respect the real-time constraint.

[1]FPS: Frame per Second.

[2]60FPS means two frames are separated by $16.67ms$. Plus, only 10% of the processing budget are granted for AI, hence the threshold $1.67ms$.

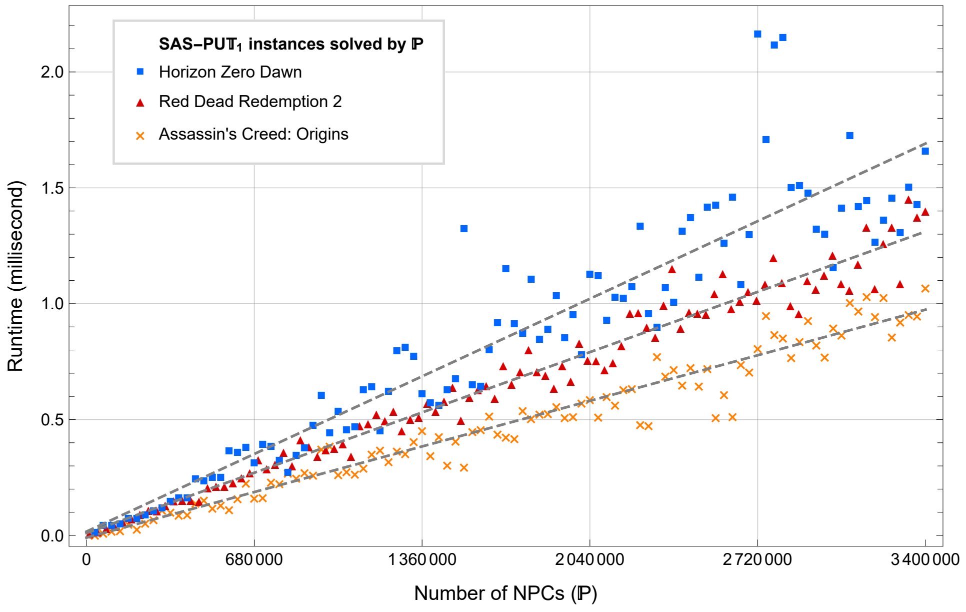

Fig. 3. The performance of $\mathbb{P}$ while solving RDR-2, ACO and HZD.

III. Discussion

In current commercial video games, as mentioned in the introduction, the number of calls to the planner is explicitly limited by studios in order to stay within the allocated processing budget. In addition, they try to keep the plans short so that the planner does not waste too much time building them. To this end, some actions are just animations, like Weapon sheathing in Shadow of Mordor [10], so that these “fake” actions do not lengthen the plans while making the player believes they were planned. This strengthens our claim on the MoveTo actions which are related to pathfinding and thus can burden planning. In our opinion, pathfinding can be done separately from planning, for example during an animation or while executing a plan.

Our benchmarks show that it is possible to represent realistic problems with the class SAS-PUT$\mathbb{T}_1$. The creation of actions can be tricky, however, as some concessions may be required to stay within the class of problems. For example with our `Horse Breeder` problem, if we want a plan with both Fill Horse Feeder and Fill Horse Trough, then our representation does not enable the horse breeder to fill the feeder before the trough. To achieve that, our model must be changed, or we can create smaller sub-goals, like $s_\star(\text{FillFeeder})$ and $s_\star(\text{FillTrough})$, so that the horse breeder can fill the feeder before the trough by achieving the goal $s_\star(\text{FillFeeder})$ before $s_\star(\text{FillTrough})$.

Concerning the results, one of the reasons explaining the runtime performance of $\mathbb{P}$ is the pre-processing, or setup. We have setup a very rigorous memory management (*malloc*, *memset*, *hashing table*), and we have allocated and filled data structures as much in advance so as to not lose time while planning. This setup helped to get the smooth curves in Figure 3. Then, our C++ implementation of $\mathbb{P}$ uses pointers and references, i.e. it plans with light objects, and this added to the calibration of the memory space made during the setup certainly played a role in the performance of $\mathbb{P}$.

IV. Conclusion

Our objective is to provide a planner for video games that is able to provide an action plan for as many NPCs as required in real-time. To this end, our approach is to model NPCs and their actions as SAS-PU$\mathbb{T}_1$ problems in order to solve the instances of these problems with the linear-time algorithm $\mathbb{P}$.

We have made two main contributions in this paper. First, the benchmarks we present show that NPCs of existing commercial video games, namely Red Dead Redemption, Assassin’s Creed: Origins and Horizon Zero Dawn, can indeed be modeled as SAS-PU$\mathbb{T}_1$ problems. Second, the results of our experiments with these benchmarks show that an optimized implementation of $\mathbb{P}$ is capable of real-time planning for millions of NPCs, with plans having up to 10 actions. This finally highlights that planning in real-time is feasible with $\mathbb{P}$ on CPU and at large scale with current technologies.

Our future work will be to consider the SAS*-structure which allows the use of undefined values in the goal state [37]. The definition of goals with a SAS-structure can be delicate as every state variable must be defined. With the SAS*-structure, however, their definition will be eased. Lastly, although our benchmarks are based on commercial video games, our experiments were performed on abstract setting. An evident future work is thus to integrate $\mathbb{P}$ into a handcrafted game or into a commercial video game that allows modding.

References

[1] J. Orkin, “Agent architecture considerations for real-time planning in games,” in *Proceedings of the* 1^{st} *AIIDE*, 2005, pp. 105–110.

[2] ——, “Three States and a Plan: The A.I. of F.E.A.R.” in *Proceedings of the Game Developer Conference*, 2006, p. 17 pages.

[3] J. Ocampo, "F.E.A.R. review," https://www.gamespot.com/reviews/fear-review/1900-6169771/, April 2007, accessed on December 3^{rd}, 2021.

[4] D. Hillburn, *Simulating Behavior Trees – A Behavior Tree/Hybrid Planner Approach*. CRC Press, 2013, vol. 1, ch. 8, pp. 99–111.

[5] T. Humphreys, *Exploring HTN Planners through Examples*. CRC Press, 2013, vol. 1, ch. 12, pp. 149–167.

[6] R. Straatman, T. Verweij, A. Champandard, R. Morcus, and H. Kleve, *Hierarchical AI for Multiplayer Bots in Killzone 3*. CRC Press, 2013, ch. 29, pp. 377–390.

[7] W. van der Sterren, *Hierarchical Plan-Space Planning for Multi-Unit Combat Maneuvers*. CRC Press, 2013, vol. 1, ch. 13, pp. 169–183.

[8] Monolith Productions, "F.E.A.R. public tools," June 2006.

[9] S. Horti, "Why F.E.A.R.'s AI is still the best in first-person shooters – Flank, cover and run away," https://www.rockpapershotgun.com/why-fears-ai-is-still-the-best-in-first-person-shooters, April 2017, accessed on December 3^{rd}, 2021.

[10] P. Higley, "GOAP at monolith productions," GDC AI Summit, March 2015.

[11] C. Conway, "GOAP in Tomb Raider," GDC AI Summit, March 2015.

[12] S. Girard, "Postmortem: AI action planning on Assassin's Creed Odyssey and Immortals Fenyx Rising," https://www.gamedeveloper.com/programming/postmortem-AI-action-planning-on-Assassins-Creed-Odyssey-and-Immortals-FenyxRising-, November 2021, accessed on december 1st 2021.

[13] M. van der Leeuw, "The PS3's SPU in the real world – a Killzone 2 case study," Game Developer Conference, March 2009.

[14] A. Champandard, T. Verweij, and R. Straatman, "Killzone 2 multiplayer bots," in *Paris Game AI Conference*, AIGameDev. https://www.guerrilla-games.com/read/killzone-2-multiplayer-bots (accessed on December 1^{st}, 2021), June 2009.

[15] S. Hollister, "The matrix awakens didn't blow my mind, but it convinced me next-gen gaming is nigh," https://www.theverge.com/22828860/the-matrix-awakens-ps5-xbox-series-x-free-next-gen, december 2021, accessed on January 12^{th}, 2022.

[16] S. Cardon and r. Jacopin, "Binary GPU-planning for thousands of NPCs," in *IEEE Conference on Games*. IEEE Press, August 2020, pp. 678–681.

[17] T. Bylander, "Complexity results for planning," in *Proceedings of* 12^{th} *IJCAI*, August 1991, pp. 274–279.

[18] K. Erol, D. Nau, and V. Subrahmanian, "Complexity, decidability and undecidability results for domain-independent planning," *Artificial Intelligence*, vol. 76, no. 1-2, pp. 75–88, 1995.

[19] C. Domshlak and R. Brafman, "Structure and complexity in planning with unary operators," in *Proceedings of the* 6^{th} *International Conference on Artificial Intelligence Planning Systems*. AAAI Press, 2002, pp. 34–43.

[20] C. Bäckström and B. Nebel, "Complexity results for SAS planning," *Computational Intelligence*, vol. 11, no. 4, pp. 625–655, 1995.

[21] É. Jacopin, "Game AI planning analytics: The case of three first-person shooters," in *Proceedings of the* 10^{th} *AIIDE*. AAAI Press, 2014, pp. 119–124.

[22] G. Prévost, S. Cardon, T. Cazenave, C. Guettier, and É. Jacopin, "La planification sas sous forme de tri topologique," in *CNIA 2022: Conférence Nationale en Intelligence Artificielle*, 2021.

[23] Rockstar Studios, "Red Dead Redemption 2," November 2018.

[24] Ubisoft, "Assassin's creed: Origins," https://store.ubi.com/fr/assassin-s-creed-origins-all-games, Octobre 2017, accessed on May 18^{th}, 2022.

[25] Guerilla Games, "Horizon zero dawn," https://www.guerrilla-games.com/games, December 2017, accessed on May 18^{th}, 2022.

[26] C. Bäckström, "Computational complexity of reasoning about plans," Ph.D. dissertation, Department of Computer and Information Science, Linköping University, september 1992.

[27] X. Neufeld, S. Mostaghim, D. L. Sancho-Pradel, and S. Brand, "Building a planner: A survey of planning systems used in commercial video games," *IEEE Transactions on Games*, vol. 11, no. 2, pp. 91–108, 2017.

[28] D. Chapman, "Planning for conjunctive goals," *Artificial intelligence*, vol. 32, no. 3, pp. 333–377, 1987.

[29] G. Prévost, S. Cardon, T. Cazenave, and É. Jacopin, "Planification dans les jeux-vidéos: À quoi servent les coûts des actions?" in *RJCIA 2019: Rencontres des Jeunes Chercheurs en Intelligence Artificielle*, 2019, pp. 66–68.

[30] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009, ch. VI, pp. 549–552.

[31] Take Two Interactive, "SEC Filing," https://ir.take2games.com/node/27706/html, July 2021, accessed on January 13^{th}, 2022.

[32] C. Livingston, "Red dead redemption 2 npcs lead interesting lives," https://www.pcgamer.com/red-dead-redemption-2-npcs-lead-interesting-lives/, November 2020, accessed on May 21^{th}, 2022.

[33] DefendTheHouse, "NPC Daily Life in Read Dead Redemption 2," https://www.youtube.com/watch?v=MrUJJgppMn4, November 2018, accessed on january 10^{th}, 2022.

[34] Jean-Marie Santoni Costantini, "Going Off-Script: Refactoring the NPC Mission System in Assassin's Creed: Origins," https://www.gdcvault.com/play/1025145/Going-Off-Script-Refactoring-the, 2018, Game Developer Conference.

[35] L. Vader, "Ubisoft's quest to make assassin's creed origins' world their most dynamic," https://www.gameinformer.com/b/features/archive/2017/06/28/ubisoft-s-quest-to-make-assassin-s-creed-origins-world-their-most-dynamic.aspx, June 2017.

[36] WikiFandom, "Horizon Zero Dawn wikifandom," https://horizon.fandom.com/wiki/Machine, January 2017, accessed on May 9^{rd}, 2022.

[37] P. Jonsson and C. Bäckström, "State-variable planning under structural restrictions: algorithms and complexity," *Artificial Intelligence*, vol. 100, no. 1-2, pp. 125–176, april 1998.